Energy Measurement

For the CM and ESB modules there is fine-grained energy measurement in place using Megware's energy meters, attached to the respective compute nodes. In addition, the DCDB monitoring framework is deployed on the entire DEEP-EST prototype, providing a variety of sensor measurements in addition to energy consumption. There are different ways to get information about energy consumption for your jobs (and the nodes). Preferred methods are:

SLURM sacct command

This is probably the easiest way to get energy consumption for your interactive and batch jobs. Once your job has finished, you can use the `sacct` command to enquire the Slurm database about its energy consumption. For further information and an example on how to use the command for energy measurements, please see accounting.

Using DCDB

The "Data Center Data Base" (DCDB) stores measured values from node and infrastructure sensors at a high frequency (every 10 seconds), including power and energy consumption of the compute nodes. This allows for a very fine-grained analysis of consumed energy for your jobs, e.g. by specifying precise time-stamp ranges and by leveraging access to measured data from different components like CPU, GPU or memory, instead of using only single accumulated values. Hence, it offers a convenient way for analysis of SLURM jobs.

DCDB is currently deployed on the DEEP-EST prototype and it is continuously collecting sensor data. A total of 62595 sensors are available, each with a time-to-live of 1 year. Please refer to D5.4 for the full list of monitored sensors and further details about the DCDB deployment. The DCDB Gitlab ?repository is also a good source of documentation and usage examples. Two types of data are available:

- **Sensor Data**: ordinary sensors sampled from the compute nodes or the surrounding infrastructure (e.g., power, temperature, CPU instructions).
- **Job Data**: a selected number of sensors are aggregated on a per-job basis, i.e., from all the compute nodes on which a certain job was running.

DCDB is installed on the DEEP-EST `deepv` login node, from which data can be queried. To prepare the shell environment and expose the DCDB tools, please run the following:

```
source /opt/dcdb/dcdb.bash
```

At this point DCDB data can be queried using the `dcdbquery` and `dcdbconfig` tools. A distributed Cassandra instance is running on the dp-mngt01, dp-mngt02 and dp-mngt03 machines: only these three should be queried by the DCDB tools. By default, there is no need to be specify the target host, as the DCDB tools use a pre-defined value; in case you wish to explicitly select a host to be queried, you can do so with the `-h` option.

The dcdbconfig Tool

The `dcdbconfig` tool can be used to perform a variety of metadata-related actions. Running the following command will list all available actions:

```
dcdbconfig help
```

We will present the main dcdbconfig actions that can be of use to users. To print the complete list of sensors available to be queried on the prototype, run the following command:

```
dcdbconfig sensor list
```

To show the metadata for a specific sensor name, run the following command:

```
Template: dcdbconfig sensor show <SENSORNAME>
Example:  dcdbconfig sensor show /deepest/esb/s25/MemUsed
```

Sensor names are slash-separated strings structured like MQTT topics, where each segment indicates a level of the system (e.g., `deepest` system, `esb` module, `s25` compute node). To show the list of jobs stored in the DCDB database, that are currently running on the prototype, run the following command:

```
dcdbconfig job running
```

To show the details of a specific job (e.g., node list, start time), run the following command:

```
Template: dcdbconfig job show <JOBID>
Example:  dcdbconfig job show 17435
```

A series of other actions to manage the sensor database are available, however users are advised not to use them so as not to alter the system?s behavior.

The dcdbquery Tool

The `dcdbquery` tool allows to perform sensor queries, returning results in CSV format. It requires a sensor name and a time range. To perform a basic query, run the following:

```
Template: dcdbquery <SENSORNAME> <TSTART> <TEND>
Example:  dcdbquery /deepest/cn/s01/power now-5m now
```

Instead of a single sensor, a list of space-separated sensor names can be supplied as well; these will be queried sequentially. Instead of using relative time-stamps, absolute UNIX time-stamps (in s, ms, ns) as well as time/date strings can also be used:

```
Example: dcdbquery /deepest/cn/s01/power 1596543564693158000 1596543788812750000
        dcdbquery /deepest/cn/s01/power "2020-08-04 14:19:24" "2020-08-04 14:23:08"
```

Energy and Power Sensors

DCDB provides several types of energy and power measurements that can be useful for performance analysis. Depending on the data source, these can have different units and scales - this information can be shown by querying a sensor with the `dcdbconfig sensor show` command. In detail, these are the energy sensors available for compute nodes in the DEEP-EST prototype:

- **energy** [Joules]: energy consumption of a compute node as a whole.
- **pkg-energy** [MicroJoules]: energy consumption of a compute node's CPUs.
- **dram-energy** [MicroJoules]: energy consumption of a compute node's memory components.
- **gpu0/energy** (ESB, DAM) [MilliJoules]: energy consumption of Nvidia V100 GPUs.
- **gpu0/sysfs-energy** (ESB) [Joules]: alternative sensor for the energy consumption of Nvidia V100 GPUs, not including the energy drawn via PCIe.
- **sysfs-energy** (ESB) [Joules]: energy consumption of an ESB compute node, excluding the GPU.

These, instead, are the available sensors for estimating power consumption:

- **power** [MilliWatts]: power consumption of a compute node as a whole.
- **gpu0/power** (ESB, DAM) [MilliWatts]: power consumption of Nvidia V100 GPUs.
- **gpu0/sysfs-power** (ESB) [MilliWatts]: alternative sensor for the power consumption of Nvidia V100 GPUs, not including the energy drawn via PCIe.

In the case of multi-socket systems (i.e. CN and DAM), the pkg and dram energy and power sensors are available both for each single socket (e.g., under `socket0/pkg-energy`), as well as for the entire compute node. The same aggregation scheme applies for sensors associated with CPU performance counters. Aside from those on compute nodes, many other energy and power sensors are available on the DEEP-EST prototype: these are associated with the system's infrastructure and cooling equipment.

Using dcdbquery for Job Queries

DCDB collects a series of aggregated metrics for each job running on the system, every 30s. These can be queried using the same `dcdbquery` syntax as above. The sensor name, however, should be constructed as follows:

```
Template: dcdbquery /job<JOBID>/<SENSORNAME><STATNAME> <TSTART> <TEND>
Example:  dcdbquery /job788354/pkg-energy.avg now-1h now
```

The `<JOBID>` field identifies the job to be queried. This matches with the job IDs reported by SLURM, and follows the same format of the latter for job packs and job arrays. For example, the following are valid queries:

```
Job Pack Example:  /job122444+1/pkg-energy.avg
Job Array Example: /job122999_15/pkg-energy.avg
```

In the job pack example above, data is queried for a specific sub-job in the pack. However, aggregated sensor data also exists for the pack as a whole, thus including measurements from multiple modules or allocations on the DEEP-EST prototype. This aggregated data can be queried using the base job

ID of the pack, omitting the + notation. No high-level aggregation is performed across the different jobs of a job array.

The <SENSORNAME> field identifies instead the aggregated metric and can be one of the following:

- **energy** [Joules]: energy consumption of nodes, as computed from the respective energy sensors.
- **dram-energy** [MicroJoules]: energy consumption of the nodes' memory, as computed from the dram-energy sensors.
- **pkg-energy** [MicroJoules]: energy consumption of the nodes' CPUs, as computed from the pkg-energy sensors.
- **MemUsed** [KiloBytes]: amount of used RAM on the compute nodes.
- **instructions**: number of CPU instructions executed on the compute nodes.
- **cpu-cycles**: amount of CPU cycles (affected by frequency scaling) on the compute nodes.
- **cache-misses**: total amount of CPU cache misses on the compute nodes.
- **cache-misses-l2**: amount of CPU L2 cache misses on the compute nodes.
- **cache-misses-l3**: amount of CPU L3 cache misses on the compute nodes.
- **scalar-double**: number of double precision floating point operations on the compute nodes' CPUs.
- **scalar-double-128**: number of double precision floating point operations using 128-bit vectorization.
- **scalar-double-256**: number of double precision floating point operations using 256-bit vectorization.
- **scalar-double-512**: number of double precision floating point operations using 512-bit vectorization.
- **gpu-energy** (DAM, ESB) [MilliJoules]: energy consumption of the compute nodes' GPUs, as computed from the gpu0/energy sensors.
- **ib-portXmitData** (CN, ESB) [Bytes]: amount of data transmitted over the Infiniband network.
- **ib-portRcvData** (CN, ESB) [Bytes]: amount of data received over the Infiniband network.

Finally, the <STATNAME> field identifies the actual type of aggregation performed from the readings of the queried sensor, by combining the data of all compute nodes on which the job was running. It can be one of the following:

- **.sum**: sum of all measurements of a metric, in a certain 30s time window.
- **.avg**: average of the measurements of a metric across nodes, in a certain 30s time window.
- **.med**: median of the measurements of a metric across nodes, in a certain 30s time window.

In order to get a cumulative measure of a job's performance (e.g., total energy spent or total amount of instructions computed), `.sum` should be used. The `.avg` and `.med` aggregations, on the other hand, indicate single-node performance in each 30s time window. It should be noted that, for a job to be measured by DCDB, its duration has to be longer than 30s. We currently expect to store job data for the entire lifetime of the system, and hence the time-to-live of 1 year does not apply to it.

BeeGFS Performance Metrics

DCDB also exposes a series of performance metrics associated with the BeeGFS distributed file system. Comprehensive documentation about the available metrics and their meaning is available ?here. All BeeGFS sensors in the DCDB database are identified by the `/deepest/beegfs/` prefix, and they can be listed by using the following command:

```
dcdbconfig sensor list | grep beegfs
```

Most BeeGFS sensors do not directly map to users and compute nodes in the DEEP-EST prototype, and they require knowledge of the system's architecture to be interpreted.

Long-term Sub-sampled Sensor Data

The sensor data collected by DCDB (excluding per-job data) has a time-to-live of 1 year. After this interval the data will be automatically erased from the database. However, to keep track of the system?s history, DCDB automatically computes sub-sampled versions of all sensors. These are kept forever in the database, and can be queried as follows:

```
Template: dcdbquery ?<OPNAME>(<SENSORNAME>)? <TSTART> <TEND>
Example:  dcdbquery ?.avg300(/deepest/cn/s01/power)? now-1h now
```

In this case, quoting is required to ensure proper parsing of the command in the shell. The <OPNAME> field defines the particular type of sub-sampling that is to be queried for the given sensor. The following alternatives are available:

- .avg300 (5-minute averages)
- .avg3600 (hourly averages)

DCDB computes sub-sampled versions of all sensors, with the exception of CPU core-level sensors (e.g., performance counters for specific CPU cores) and per-job sensors, to avoid inflating the size of the databases indefinitely. In general, the operations available for a DCDB sensor are shown along with its metadata when using the `dcdbconfig sensor show` command.

Grafana Visualization

A Grafana instance is also running and is reachable at `grafanavm:3000`. The `grafanavm` host is visible only from the prototype?s network, so ssh tunneling is required to reach it. In order to use Grafana, an account must be created: if you wish to have access, please contact Alessio Netti (alessio.netti@?) and one will be created for you. Normally, the Grafana instance should be already set up and you will not need to configure the DCDB data source. If that is not the case, please follow the instructions below.

Log into the Grafana web frontend and navigate into the `Configuration->Data Sources` menu. Here, click on `Add Data Source` and select DCDB. Please use the following values for the required fields in the configuration window of the DCDB data source:

```
URL: https://dp-mngt01:8081
Access: Server (Default)
User and Password: admin
Check "Basic Auth" and "Skip TLS Verify" under the "Auth" table
```

At this point click on `Save & Test` once and, if the procedure was successful, you will be ready to add dashboards and panels using DCDB data.

Using /sys files

The energy meters provide their measured values through the "/sys" filesystem on the compute nodes using different files. To query the overall energy (in Joules) a node has consumed so far, you can use the `energy_j` file. You should integrate readings in your SLURM job script before and after you `srun` your commands to measure consumed energy by your commands (applications):

Unit=[Joules]

CM Module

```
srun sh -c 'if [ $SLURM_LOCALID == 0 ]; then echo ${SLURM_NODEID}: $(cat /sys/devices/platform/sem.5/energy_j); fi'
```

ESB Module

There are two energy meters present for the ESB nodes, one for the CPU blade and one for the GPU part:

```
srun sh -c 'if [ $SLURM_LOCALID == 0 ]; then echo ${SLURM_NODEID}: $(cat /sys/devices/platform/sem.1/energy_j); fi'
srun sh -c 'if [ $SLURM_LOCALID == 0 ]; then echo ${SLURM_NODEID}: $(cat /sys/devices/platform/sem.2/energy_j); fi'
```

To get the consumed energy by a multi-node job you have to accumulate all the values which makes it quite cumbersome, but for running on single nodes (maybe even in an interactive session) reading out current values directly from the files might be quite useful.